CHEF: A STATUS REPORT *

J.-F. Ostiguy and L. Michelotti
Fermi National Accelerator Laboratory, Batavia, IL 60510, USA

Abstract

CHEF refers both to a framework and to an interactive application emphasizing accelerator optics calculations. The framework supports multiple domains of applications: e.g. nonlinear analysis, perturbation theory, and tracking. Its underlying philosophy is to provide an infrastructure with minimum hidden implicit assumptions, general enough to facilitate both routine and specialized computational tasks, and to minimize the duplication of necessary, complex bookkeeping tasks. CHEF was already described in recent conferences [1, 2]. This paper is a status report on recent developments, including issues related to applications to high energy linacs.

INTRODUCTION

CHEF is a hierarchical framework, implemented in C++, that has its roots in libraries developed at Fermilab throughout the 1990s. Just like COSY [3], FPP [4] and a few other codes, it uses as a foundation layer, an automatic differentiation engine (`mxyzptlk`, in this case). This design allows CHEF to provide unified support for conventional first order optics and higher order computations. Algorithms which in traditional codes are formulated in terms of matrices are instead, expressed in terms of `Mapping`s. Passage through an element transforms a particle's phase space state: in a conventional optics code, the transfer matrix entries can be interpreted as the first order derivatives of this transformation. In CHEF, the transformation is represented by a `Mapping`, implemented as a vector of `Jet` objects. Each `Jet` component defines a series expansion which describes the transformation undergone by one of the coordinates, up to some specified order. A first order `Mapping` and a traditional transfer matrix contain equivalent information; however, `Mapping` is a more general abstraction.

A substantial fraction of the complexity in an accelerator code has to do with bookkeeping tasks. One of the objectives of CHEF is to provide components that enable accelerator scientists to either build or add new features to applications without having to become too involved in such tasks. With that in mind, CHEF provides facilities to manipulate, transform, edit and display hierarchical beamlines, parse lattice files and perform a variety of standard calculations. Although originally geared towards synchrotrons, interest in issues associated with the International Linear Collider have motivated extensions to existing functionality and development of new capabilities. We discuss some recent developments.

* Work supported by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy.

XSIF PARSER

For a few years now, CHEF has had the ability of reading lattice descriptions in MAD8 format. The parser had a few limitations, the most important being not being able to process macro statements. To exchange lattice information, the ILC collaboration has officially settled on the XSIF format (MAD8 augmented with linac-specific extensions). In order to be usable as a tool to investigate ILC issues, CHEF needed to deal with those files in unmodified form. A new XSIF parser has therefore been implemented.

The new parser is based on the standard `flex/bison` token analyzer/parser generator combination. In contrast with their close cousins `lex` and `yacc` which are meant to be used in a C programming environment. `flex` and `bison` provide support for C++ constructs; this was taken advantage of. The parser is fully re-entrant and recognizes all the elements types and attributes of the XSIF specification, variable expressions (either immediate or deferred) as well as arbitrarily nested macros and include files. Currently, it handles all officially released ILC files.

While development was in progress, we became aware of an effort to develop a "universal" XSIF parser, using XML as a intermediate lattice exchange format [5]. Unfortunately, it was not ready for use on a time scale compatible with our needs. This said, experience shows that decoupling the parser from the internals of an accelerator code is a task that is easily underestimated.

BEAMLINES

Typically, beamlines are specified in a hierarchical manner, that is, beamlines are comprised of building blocks i.e. standard groups of elements responsible for certain specific functions e.g. a regular cell in a periodic focusing structure, a dispersion suppressor, a zero dispersion insertion, a low beta insertion, etc. A designer often starts with a periodic structure and gradually introduces various local modifications.

Beamlines are recursive structures, in the sense that a beamline is a container that may contain either atomic elements or other beamline containers. CHEF provides extensive facilities to manipulate, edit and transform hierarchical beamlines. Internally, a `beamline` is implemented as a tree where sibling nodes are stored in doubly-linked lists. Originally, the lists were custom containers of raw pointers to elements. This resulted in significant complexity, due in

part to pointer ownership ambiguity. Consider for example a function that removes elements from a beamline. Who owns these elements and who is responsible for deleting them? When is it safe to delete them? Although consistent function behavior with respect to object ownership helps, in practice some rules are violated –not necessarily consciously – resulting in subtle and unexpected side-effects. Ultimately the code becomes brittle and increasingly hard to maintain.

To alleviate these issues, CHEF was re-factored to take maximum advantage of STL containers and algorithms. To address pointer ownership issues, at each level of a beamline tree, sibling nodes are now stored in a `std::dlist` of `tr1::shared_ptr` (a reference-counted shared pointer with value semantics). Further, the beamline class has been endowed with iterators that fully model the STL reversible iterator concept: `iterator` (single level), `pre_order_iterator` (depth-first order), `post_order_iterator` (breath-first order), `deep_iterator` (iterates recursively through elements only). All iterators are also available in the `reverse` and `const` varieties for a total of sixteen distinct iterator types. Aside from the fact that this provides a familiar and consistent interface, using STL compatible iterators has the distinct advantage of making STL implementation of standard algorithms automatically available for beamline containers.

CHEF makes a large fraction of its interface available to python scripts through bindings based on boost::python, a compile-time template metaprogram bindings generator. boost:python provides standard infrastructure to map STL compliant iterators into native python iterators with minimal effort.

PARTICLE BUNCHES

Emittance preservation studies imply a capability to propagate particle bunches. While such capability has existed in CHEF for some time, it was rudimentary and not designed to be efficient in the context of large scale emittance preservation studies involving statistical studies of misalignment, ground motion and so forth.

The existing `ParticleBunch` class has been improved. A `ParticleBunch` acts as container for (abstract) `Particle` objects. It can be populated, based on standard beam parameters, according to different standard statistical distributions. The implementation is based on a `boost::ptr_vector` container. This container behaves like a standard `std::vector` container, with some interesting distinctions. In contrast with a standard container, it is designed to hold exclusively owned raw pointers to dynamically allocated `Particle` objects, rather than the objects themselves. Nevertheless, its iterators, when dereferenced, return references to the actual objects, not pointers. One important consequence of holding pointers is that mutating algorithms, in particular sorting, can be performed much more efficiently since it is more efficient to re-order pointers to objects rather that the objects themselves.

Since propagating a bunch through a sequence of accelerator components involves repeatedly and on a very large scale, iterating through all particles contained in a bunch, locality of `Particle` object references is important to ensure good performance. For this reason, `Particle` objects are sequentially allocated in memory from a private pool owned by each `ParticleBunch` object.

CHEF supports all the expected concrete particle types i.e. `Proton`, `Electron`, `Positron`, etc. Each of these types is derived from an abstract `Particle` type which is never (and cannot be) explicitly instantiated. One consequence of the design described above is that although a `ParticleBunch` has no explicit knowledge of the concrete type of the `Particles` it contains, the specific run-time identities of `Particles` is preserved since they are not held by value.

One important aspect of support for bunches is to provide facilities to generate macroscopic projections. The need for such projections arises, for example, in the modeling of beam diagnostic instruments. In the context of high energy linacs, an important issue is the simulation of wakefield effects as described in more detail below. For the purpose of computing wakefields, a bunch is typically described in terms of a longitudinal charge or transverse dipole moment distribution. To obtain this distribution, it is necessary to (1) sort particles according to their longitudinal position (2) assign them to a finite number of longitudinal bins (3) for each bin, compute the relevant quantity. In CHEF, this functionality is provided by an auxiliary `BunchProjector` helper class whose constructor is passed a reference to a `ParticleBunch`. Instantiation of a `BunchProjector` results in the bunch being sorted longitudinally and projections computed. It is worth pointing out that in a high energy linac, while the longitudinal ordering of particles within a bunch is usually preserved, this cannot be assumed to be always true (for example, after going through a bunch compressor). To insure good performance, the sorting algorithm should be able to handle nearly ordered bunches efficiently.

WAKEFIELDS

Emittance preservation is a critical issue in linear colliders. Wakefields – the electromagnetic fields induced by the reaction of the environment to the beam while it passes through accelerator elements and, in particular, accelerating cavities – contribute significantly to emittance growth. It is convenient to consider separately two kinds of wakefields: (1) long range wakefields, the fields induced by downstream bunches and (2) short-range wakes, the fields induced by the particles within a given bunch and acting back on it. Modeling either kind of wake involves similar techniques; the differences lie in the level of detail needed to describe a bunch and the need to keep track of previous history. In simulations involving a superconducting linac, long range wakes are often omitted from simulations because they arise from high order accelerating cavity modes

which are extracted and damped with special absorbers. Short range wakes, on the other hand, cannot be neglected.

Assuming cylindrical symmetry, it can be shown the net impulse on a point particle located at a position z within a bunch involves evaluating a convolution integral of the form

$$\Delta p(z) = \int_{-\infty}^{\infty} W(z - z')g(z')dz' \qquad (1)$$

where $g(z)$ represents some suitable longitudinal density. Assuming that $g(z)$ and the wake $W(z)$ are discretized over N intervals, numerical integration of the above involves $O(N^2)$ operations. While straightforward to implement, naive numerical integration is an inefficient way of computing a convolution. Since a high energy linac such as the proposed ILC would be comprised of 10^3 accelerating structures, wakefield computations become a bottleneck in the context of dynamic or statistical studies.

By taking advantage of the fact that convolution is equivalent to a multiplication in Fourier space the integral in Equation 1 can be computed in $O(N \log N)$ operations using FFTs. This approach was adopted for the implementation of short-range wakefields in CHEF.

It is important to note that the FFT of $g(z)$ is not the Fourier transform of g; rather, it is the Fourier transform of the periodic extension of g. In practice both $g(z)$ and $W(z)$ are non-zero over finite intervals. No aliasing error will be introduced provided that sampling is performed over an interval which, at minimum, is equal to twice the maximum of the supports of g and W.

Internally, CHEF provides an element (`WakeKick`), that acts exclusively on bunches. While this element can in principle be used as a standalone element, it is meant to be used as a building block for more complex composite elements. For example, a linear accelerator superconducting structure is implemented as a sequence of two substructures with a `WakeKick` element inserted in the middle. Wakes can be described either analytically or in table form. Note that for the duration of a simulation, the wake $W(z)$ remains unchanged. To take advantage of this fact and avoid repeatedtly computing the same wake FFT, convolution is implemented as a stateful `ConvolutionFunctor` function object with shared semantics. Concretely, this means that once a `WakeKick` element is instantiated for a particular wake, it can be copied multiple times and each instance will automatically share a single copy of the `ConvolutionFunctor` implementation. Cavities not required to share the same wake.

DISPERSION COMPUTATIONS

Since CHEF contains an automatic differentiation engine, the infrastructure is available to compute dispersion accurately without the need to resort to explicit finite differencing. Nevertheless, until recently, dispersion in CHEF was computed conventionally, that is, by computing the difference between the trajectories of tracking two particles separated in momentum by some small amount dp/p and by taking the difference between trajectories. CHEF can now compute dispersion directly using `JetParticles`.

In the case of a linac, the implementation is straightforward: a `JetParticle` with suitable initial conditions is propagated: the dispersion is automatically available as the term correponding to the first order derivative with respect to the state variable dp/p_0. For a ring, the situation is somewhat more complex, because the dispersion now represents the change in periodic orbit associated with a change in momentum. The algorithm involves iteratively solving for a periodic orbit in `Jet` space.

Aside from the fact that the computation is accurate to machine precision, all coupling terms are also immediately available. Finally, if desired, the dispersion can be obtained to any specified order.

PERFORMANCE

Unification of first order optics and higher order computations comes at a cost. Generality implies increased code complexity and tends to lead to lower performance: software with a narrower scope can rely on a-priori simplifying assumptions. Following a substantial overhaul of `mxyzpltk`, the automatic differentiation engine, overall performance for standard first order computations now compares very favorably to matrix-based codes. The details are the object of a companion report [6].

CONCLUSION AND FUTURE PLANS

Interest in simulating high energy linacs has and continues to stimulate development of CHEF. In the near future, we plan to add a capability to model undulators such as those envisioned in the ILC polarized positron source and study their impact on emittance preservation. We also expect to continue optimizing performance to facilitate large scale parametric studies involving misaligments and ground motion.

REFERENCES

[1] L. Michelotti, J.-F. Ostiguy, "CHEF: An Interactive Program for Accelerator Optics Calculations", Proc. PAC, p 988 (2005)

[2] J.-F. Ostiguy, L. Michelotti, "CHEF: An Interactive Program for Accelerator Optics Calculations", Proc. ICAP 2006, Chamonix, France

[3] Makino, M. Berz, COSY Infinity, Version 9, Nuclear Instruments and Methods A558 (2005), pp. 346-350

[4] E. Forest, F. Schmidt, "The Full Polymorphic Package" , ACM SIGPLAN Fortran Forum, 20,3 (December 2001)

[5] D.Sagan, D.A. Bates and A. Wolski, "The Universal Accelerator Parser" , Proc. ICAP 2006, Chamonix, France

[6] J.-F. Ostiguy, L. Michelotti, "Mxyzptlk: An efficient, Native C++ Automatic Differentiation Engine", this conference.